How to Explain Empirical Distribution of Software Defects by Severity

Francisco Zapata[1], Olga Kosheleva[2], and Vladik Kreinovich[3]

[1] Department of Industrial, Manufacturing, and Systems Engineering
University of Texas at El Paso, El Paso, TX 79968, USA
fazg74@gmail.com
[2] Department of Teacher Education
University of Texas at El Paso, El Paso, TX 79968, USA
olgak@utep.edu
[3] Department of Computer Science
University of Texas at El Paso, El Paso, TX 79968, USA
vladik@utep.edu

Abstract. In the last decades, several tools have appeared that, given a software package, mark possible defects of different potential severity. Our empirical analysis has shown that in most situations, we observe the same distribution or software defects by severity. In this paper, we present this empirical distribution, and we use interval-related ideas to provide an explanation for this empirical distribution.

Keywords: software defects, severity distribution, empirical distribution, theoretical explanation, interval techniques

1 Empirical Distribution of Software Defects by Severity

Automatic detection and classification of defects. Software packages have defects of different possible severity. Some defects allow hackers to enter the system and can thus, have a potentially high severity. Other defects are minor and maybe not be worth the effort needed to correct them. For example, if we declare a variable which is never used (or we declare an array of too big size, so that most of its elements are never used), this makes the program not perfect, but does not have any serious negative consequences other than wasting some computer time on this declaration and wasting some computer memory; see, e.g., [1–4, 7, 11–14].

In the last decades, several tools have appeared that, given a software package, mark possible defects of different potential severity; see, e.g., [9].

Usually, software defects which are worth repairing are classified into three categories by their relative severity [9]:

- software defects of very high severity (they are also known as *critical*);
- software defects of high severity (they are also known as *major*); and
- software defects of medium severity.

This is equivalent to classifying *all* the defects into four categories, where the fourth category consists of *minor* defects, i.e., defects which are not worth repairing.

Cautious approach. The main objective of this classification is not to miss any potentially serious defects. Thus, in case of any doubt, a defect is classified into the most severe category possible.

As a result, the only time when a defect is classified into medium severity category is when we are absolutely sure that this defect is not of high or of very high severity. If we have any doubt, we classify this defect as being of high or very high severity.

Similarly, the only time when a defect is classified as being of high severity is when we are absolutely sure that this defect is not of very high severity. If there is any doubt, we classify this defect as being of very high severity.

In particular, in situations in which we have no information about severity of different defects, we should classify all of them as of very high severity. As we gain more information about the consequences of different defects, we can start assigning some of the discovered defects to medium or high severity categories. However, since by default we classify a defect as having high severity:

- the number of defects classified as being of very high severity should still be the largest,
- followed by the number of defects classified as being of high severity,
- and finally, the number of defects classified as being of medium severity should be the smallest of the three.

Empirical results. Since software defects can lead to catastrophic consequences, it is desirable to learn as much as possible about different defects. In particular, it is desirable to know how frequently one meets defects of different severity.

To find out the empirical distribution of software defects by severity, it is necessary to study a sufficiently large number of detects. This is a challenging task: while many large software packages turn out to have defects, even severe defects, such defects are rare, and uncovering each severe defect in a commercially used software is a major event.

At first glance, it may seem that, unless we consider nor-yet-released still-being tested software, there is no way to find a sufficient number of defects to make statistical analysis possible. However, there is a solution to this problem: namely, legacy software, software that was written many years ago, when the current defect-marking tools were not yet available. Legacy software works just fine – if it had obvious defects, they would have been noticed during its many years of use. However, it works just fine only in the original computational environment, and when the environment changes, many hidden severe defects are revealed.

In some cases, the software package used a limited size buffer that was sufficient for the original usage, but when the number of users increases, the buffer becomes insufficient to store the jobs waiting to be performed. Another typical situation is when a C program, which should work for all computers, had some

initial bugs that were repaired by hacks that explicitly took into account that in those days, most machines used 32-bit words. As a result, when we run the supposedly correct program on a 64-but machine, we get many errors.

Such occurrences are frequent. As a result, when hardware is upgraded – e.g., from 32-bit to 64-bit machines – software companies routinely apply the defect-detecting software packages to their legacy code to find and repair all the defects of very high, high, and medium severity. Such defects are not that frequent, but even if in the original million-lines-of codes software package, 99.9% of the lines of code are absolutely flawless, this still means that there may be a thousand of severe defects. This is clearly more than enough to provide a valid statistical analysis of distribution of software defects by severity.

So, at first glance, we have a source of defects. However, the problem is that companies – naturally – do not like to brag about defects in their code – especially when a legacy software, software used by many customers, turns out to have thousands of severe defects. On the other hand, companies are interested in knowing the distribution of the defects – since this would help them deal with these defects – and not all software companies have research department that would undertake such an analysis. In view of this interest, we contacted software folks from several companies who allowed us to test their legacy code and record the results – on the condition that when we publish the results, we would not disclose the company names. (We are not even allowed to disclose which defect-detecting tool was used – because many companies use their own versions of such tools.)

Interestingly, for different legacy software packages, as long as they were sufficiently large (and thus containing a sufficiently large number of severe defects), the distribution of defects by severity was approximately the same. In this paper, we illustrate this distribution on three typical cases. These cases, sorted by the overall number of defects, are presented in Table 1.

Table 1. Defects of different severity: three typical cases

	Case 1	Case 2	Case 3
Total number of defects	996	1421	1847
Very high severity defects	543	738	1000
High severity defects	320	473	653
Medium severity defects	133	210	244

Analysis of the empirical results: general case. In all the cases, the numbers of very high, high, and medium severity defects can be approximately described by the ratio $5 : 3 : 1$. In other words:

– the proportion of software defects of very high severity is close to

$$\frac{5}{5+3+1} = \frac{5}{9} \approx 56\%;$$

– the proportion of software defects of high severity is close to

$$\frac{3}{5+3+1} = \frac{3}{9} \approx 33\%;$$

and
– the proportion of software defects of medium severity is close to

$$\frac{1}{5+3+1} = \frac{1}{9} \approx 11\%.$$

Let us show it on the example of the above three cases.

Case 1. In this case, $\frac{1}{9} \cdot 996 = 110\frac{2}{3}$, so we should:

– observe $\frac{1}{9} \cdot 996 = 110\frac{2}{3} \approx 111$ medium severity defects;
– observe $\frac{3}{9} \cdot 996 = 3 \cdot \left(\frac{1}{9} \cdot 996\right) = 3 \cdot 110\frac{2}{3} = 332$ high severity defects, and
– observe $\frac{5}{9} \cdot 996 = 5 \cdot \left(\frac{1}{9} \cdot 996\right) = 5 \cdot 110\frac{2}{3} = 553\frac{1}{3} \approx 553$ very high severity defects.

As we can see from Table 2, the actual numbers of defects of different severity are very close to these numbers. The match is up to 20% accuracy, which for the problem of predicting number of software defects is not so bad.

Table 2. Case 1: comparing actual and predicted numbers of defects of different severity

	actual number	predicted number
Very high severity defects	543	553
High severity defects	320	332
Medium severity defects	133	111

Case 2. In this case, $\frac{1}{9} \cdot 1421 = 157\frac{8}{9}$, so we should:

– observe $\frac{1}{9} \cdot 1421 = 157\frac{8}{9} \approx 158$ medium severity defects;
– observe $\frac{3}{9} \cdot 1421 = 3 \cdot \left(\frac{1}{9} \cdot 1421\right) = 3 \cdot 157\frac{8}{9} = 471\frac{8}{3} \approx 474$ high severity defects, and
– observe $\frac{5}{9} \cdot 1421 = 5 \cdot \left(\frac{1}{9} \cdot 1421\right) = 5 \cdot 157\frac{8}{9} = 785\frac{40}{9} \approx 789$ very high severity defects.

Table 3. Case 2: comparing actual and predicted numbers of defects of different severity

	actual number	predicted number
Very high severity defects	738	789
High severity defects	473	474
Medium severity defects	210	158

As we can see from Table 3, the actual numbers of defects of different severity are very close to these numbers. The match is also up to ≈20% accuracy.

Case 3. In this case, $\frac{1}{9} \cdot 1847 = 205\frac{2}{9}$, so we should:

- observe $\frac{1}{9} \cdot 1847 = 205\frac{2}{9} \approx 205$ medium severity defects;
- observe $\frac{3}{9} \cdot 1847 = 3 \cdot \left(\frac{1}{9} \cdot 1847\right) = 3 \cdot 205\frac{2}{9} = 615\frac{6}{9} \approx 616$ high severity defects, and
- observe $\frac{5}{9} \cdot 1847 = 5 \cdot \left(\frac{1}{9} \cdot 1847\right) = 5 \cdot 205\frac{2}{9} = 1025\frac{10}{9} \approx 1026$ very high severity defects.

As we can see from Table 4, the actual numbers of defects of different severity are very close to these numbers. The match is also up to 20% accuracy.

Table 4. Case 3: comparing actual and predicted numbers of defects of different severity

	actual number	predicted number
Very high severity defects	1000	1026
High severity defects	653	616
Medium severity defects	244	206

2 How to Explain the Empirical Distribution

What we want: a brief reminder. We want to find the three frequencies:

- the frequency p_1 of defects of medium severity;
- the frequency p_2 of defects of high severity, and
- the frequency p_3 of defects of very high severity.

All we know is that $p_1 < p_2 < p_3$.

What we do. We will use ideas related to interval uncertainty and interval computations (see, e.g., [6, 8, 10]) to explain the above empirical dependence.

First idea: let us use intervals instead of exact numbers. In principle, these frequencies can somewhat change from one example to another – as we have seen in the above examples. So, instead of selecting single values p_1, p_2, and p_3, we should select three *regions* of possible values, i.e., we should select:

- an interval $\left[\underline{F}_1, \overline{F}_1\right]$ of possible values of p_1;
- an interval $\left[\underline{F}_2, \overline{F}_2\right]$ of possible values of p_2; and
- an interval $\left[\underline{F}_3, \overline{F}_3\right]$ of possible values of p_3.

To guarantee that $p_1 < p_2$, we want to make sure that every value from the first interval $\left[\underline{F}_1, \overline{F}_1\right]$ is smaller than or equal to any value from the second interval $\left[\underline{F}_2, \overline{F}_2\right]$. To guarantee this, it is sufficient to require that the largest value $\overline{F}_1$ from the first interval is smaller than or equal to the smallest value of the second interval: $\overline{F}_1 \le \underline{F}_2$.

Similarly, to guarantee that $p_2 < p_3$, we want to make sure that every value from the second interval $\left[\underline{F}_2, \overline{F}_2\right]$ is smaller than or equal to any value from the third interval $\left[\underline{F}_3, \overline{F}_3\right]$. To guarantee this, it is sufficient to require that the largest value $\overline{F}_2$ from the first interval is smaller than or equal to the smallest value of the third interval: $\overline{F}_2 \le \underline{F}_3$.

First idea expanded: let us make these intervals as wide as possible. We decided to have intervals of possible values of p_i instead of exact values of the frequencies. To fully follow this idea, let us make these intervals as wide as possible, i.e., let us make sure that it is not possible to increase one of the intervals without violating the above inequalities.

This means that we should have no space left between $\overline{F}_1$ and $\underline{F}_2$ – otherwise, we can expand either the first or the second interval. We should therefore have

$$\overline{F}_1 = \underline{F}_2.$$

Similarly, we should have no space left between $\overline{F}_2$ and $\underline{F}_3$ – otherwise, we can expand either the second or the third interval. We should therefore have

$$\overline{F}_2 = \underline{F}_3.$$

Also, we should have $\underline{F}_1 = 0$ – otherwise, we can expand the first interval.

As a result, we get the division of the interval $[0, F]$ of possible frequencies into three sub-intervals:

- the interval $\left[0, \overline{F}_1\right]$ of possible values of the frequency p_1;
- the interval $\left[\overline{F}_1, \overline{F}_2\right]$ of possible values of the frequency p_2; and
- the interval $\left[\overline{F}_2, F\right]$ of possible values of the frequency p_3.

Second idea: since we have to reason to take intervals of different widths, let us take them equal. We have no a priori reason to assume that the three intervals have different widths. Thus, it is reasonable to assume that these three intervals have the exact same width, i.e., that

$$\overline{F}_1 = \overline{F}_2 - \overline{F}_1 = F - \overline{F}_2.$$

From the equality $\overline{F}_2 - \underline{F}_1 = \overline{F}_1$, we conclude that $\overline{F}_2 = 2\overline{F}_1$. Now, from the condition that $F - \overline{F}_2 = \overline{F}_1$, we conclude that

$$F = \overline{F}_2 + \overline{F}_1 = 2\overline{F}_1 + \overline{F}_1 = 3\overline{F}_1.$$

So, we have the following three intervals:

- the interval $\left[0, \overline{F}_1\right]$ of possible values of the frequency p_1;
- the interval $\left[\overline{F}_1, 2\overline{F}_1\right]$ of possible values of the frequency p_2; and
- the interval $\left[2\overline{F}_1, 3\overline{F}_1\right]$ of possible values of the frequency p_3.

Third idea: which value from the interval should we choose. We would like to select a single "typical" value from each of the three intervals.

If we know the probability of different values from each interval, we could select the average value. We do not know these probabilities, so to use this approach, we need to select one reasonable probability distribution on each interval.

A priori, we have no reason to believe that some values from a given interval are more probable than others. Thus, it is reasonable to conclude that all the values within each interval are equally probable – i.e., that on each of the three intervals, we have a uniform distribution.

Comment. This conclusion can be viewed as a particular case of *Laplace Indeterminacy Principle* – and of its natural generalization, the Maximum Entropy approach; see, e.g., [5].

Now, we are ready to produce the desired probabilities. For the uniform distribution on an interval, the mean value, as one can clearly check, is the midpoint of the interval. So:

- as the estimate for p_1, we select the midpoint of the first interval $\left[0, \overline{F}_1\right]$, i.e., the value
$$p_1 = \frac{0 + \overline{F}_1}{2} = \frac{\overline{F}_1}{2};$$
- as the estimate for p_2, we select the midpoint of the second interval $\left[\overline{F}_1, 2\overline{F}_1\right]$, i.e., the value
$$p_2 = \frac{\overline{F}_1 + 2\overline{F}_1}{2} = 3 \cdot \frac{\overline{F}_1}{2};$$
- finally, as the estimate for p_1, we select the midpoint of the third interval $\left[2\overline{F}_1, 3\overline{F}_1\right]$, i.e., the value
$$p_3 = \frac{2\overline{F}_1 + 3\overline{F}_1}{2} = 5 \cdot \frac{\overline{F}_1}{2}.$$

Conclusion. We see that $p_2 = 3p_1$ and $p_3 = 5p_1$. So, we indeed have an explanation for the empirical ratios $1 : 3 : 5$ between the frequencies of software flaws of different severity.

3 What If We Had a More Detailed Classification – into More than Three Severity Levels?

Formulation of the problem. In the above analysis, we used the usual subdivision of all non-minor software defects into three levels: of medium severity, of high severity, and of very high severity. However, while such a division is most commonly used, some researchers and practitioners have proposed a more detailed classification, when, e.g., each of the above three levels is further subdivided into sub-categories.

To the best of our knowledge, such detailed classification have not yet been massively used in software industry. Thus, we do not have enough data to find the empirical distribution of software defects by sub-categories. However, while we do not have the corresponding empirical data, we can apply our theoretical analysis to come up with reasonable values of expected frequencies. Let us do it.

Analysis of the problem. We consider the situation when each of the three severity levels is divided into several sub-categories. Let us denote the number of sub-categories in a level by k. Then, overall, we have $3k$ sub-categories of different level of severity.

Similar to the previous section, it is reasonable to conclude that each of these categories correspond to intervals of possible probability values:

$$\left[0, \overline{F}_1\right], \left[\overline{F}_1, \overline{F}_2\right], \left[\overline{F}_2, \overline{F}_3\right], \ldots, \left[\overline{F}_{3k-1}, \overline{F}_{3k}\right].$$

Also, similarly to the previous section, it is reasonable to require that all these intervals are of the same length. Thus, the intervals have the form

$$\left[0, \overline{F}_1\right], \left[\overline{F}_1, 2\overline{F}_1\right], \left[2\overline{F}_1, 3\overline{F}_1\right], \ldots, \left[(3k-1)\cdot\overline{F}_1, (3k)\cdot\overline{F}_1\right].$$

As estimates for the corresponding frequencies, similar to the previous section, we take midpoints of the corresponding intervals:

$$\frac{\overline{F}_1}{2}, \quad 3\cdot\frac{\overline{F}_1}{2}, \quad 5\cdot\frac{\overline{F}_1}{2}, \ldots, \quad \frac{6k-1}{2}\cdot\frac{\overline{F}_1}{2}.$$

Results of the analysis. This analysis shows that the frequencies of detects of different levels of severity follow the ratio $1 : 3 : 5 : \ldots : (6k-1)$.

This result consistent with our previous findings. If for each of three main severity levels, we combine all the defects from different sub-categories of this level, will we still get the same ratio $1 : 3 : 5$ as before?

The answer is "yes". Indeed, by adding up the k sub-categories with lowest severity, we can calculate the total frequency of medium severity defects as

$$(1+3+5+\ldots+(2k-5)+(2k-3)+(2k-1))\cdot\frac{\overline{F}_1}{2}.$$

In the above sum of k terms, adding the first and the last terms leads to $2k$; similarly, adding the second and the last but one terms lead to $2k$, etc. For each pair, we get $2k$. Out of k terms, we have $\frac{k}{2}$ pairs, so the overall sum is equal to

$$1+3+5+\ldots+(2k-5)+(2k-3)+(2k-1)=\frac{k}{2}\cdot(2k)=k^2.$$

Thus, the total frequency of medium severity defects is $k^2\cdot\frac{\overline{F}_1}{2}$.

Similar, the total frequency of high severity defects is equal to

$$((2k+1)+(2k+3)+\ldots+(4k-3)+(4k-1))\cdot\frac{\overline{F}_1}{2},$$

where similarly, the sum is equal to

$$(2k+1)+(2k+3)+\ldots+(4k-3)+(4k-1)=\frac{k}{2}\cdot(6k)=3k^2.$$

Thus, the total frequency of high severity defects is $3k^2\cdot\frac{\overline{F}_1}{2}$.

The total frequency of very high severity defects is similarly equal to

$$((4k+1)+(4k+3)+\ldots+(6k-3)+(6k-1))\cdot\frac{\overline{F}_1}{2},$$

where similarly, the sum is equal to

$$(4k+1)+(4k+3)+\ldots+(6k-3)+(6k-1)=\frac{k}{2}\cdot(10k)=5k^2.$$

Thus, the total frequency of high severity defects is $5k^2\cdot\frac{\overline{F}_1}{2}$.

These frequencies indeed follow the empirical ratio $1:3:5$, which means that our analysis is indeed consistent with our previous findings.

What if we have 4 or more original severity levels? Another possible alternative to the usual 3-level scheme is to have not 3, but 4 or more severity levels. In this case, if we have L levels, then a similar analysis leads to the conclusion that the corresponding frequencies should follow the ratio

$$1:3:5:\ldots:(2L-1).$$

Acknowledgments

This work was supported in part by the US National Science Foundation grant HRD-1242122.

The authors are greatly thankful to the anonymous referees for their valuable suggestions and to Dan Tavrov for his help.

References

1. Ackerman A. F., Buchwald L. S., Lewski F. H.: Software inspections: an effective verification process, IEEE Software 6(3), 31–36 (1989).
2. D'Ambros M., Bacchelli A., Lanza M.: On the impact of design flaws on software defects, Proceedings of the 10th International Conference on Quality Software QSIC'2010, Zhangjiajie, China, July 14–15, 2010, pp. 23–31 (2010).
3. Harter D. E., Kemerer C. F., Slaughter S. A.: Does software process improvement reduce the severity of defects? A longitudinal field study. IEEE Transactions on Software Engineering 38(4), 810–827 (2012).
4. Javed T., Durrani Q. S.: A study to investigate the impact of requirements instability on software defects, ACM SIGSOFT Software Engineering Notes 29(3), 1–7 (2004).
5. Jaynes E. T., Bretthorst G. L.: Probability theory: the logic of science, Cambridge University Press, Cambridge, UK (2003).
6. Jaulin L., Kiefer M., Didrit O., Walter E.: Applied interval analysis, with examples in parameter and state estimation, robust control, and robotics, Springer, London (2001).
7. Kapur P. K., Kumar A., Yadav K., Khatri S. K.: Software reliability growth modelling for errors of different severity using change point. International Journal of Reliability, Quality and Safety Engineering 14(4), 311–326 (2007).
8. Mayer G.: Interval analysis and automatic result verification, de Gruyter, Berlin (2017).
9. Mitre Corp.: Common weakness enumerication: a community-developed list of software weakness types, https://cwe.mitre.org/cwss/cwss_v1.0.1.html, accessed on March 2, 1018.
10. Moore R. E., Kearfott R. B., Cloud M. J.: Introduction to interval analysis, SIAM, Philadelphia (2009).
11. Ostrand, T. J., Weyuker, E. J.: The distribution of faults in a large industrial software system, ACM SIGSOFT Software Engineering Notes 27(4), 55–64 (2002).
12. Sullivan M., Chillarege R.: Software defects and their impact on system availability – a study of field failures in operating systems, Proceedings of the 21st International Sympoisum on Fault-Tolerant Computing, Montreal, Canada, June 25–27, 1991, pp. 2–9 (1991).
13. Sullivan M., Chillarege R.: A comparison of software defects in database management systems and operating systems, Proceedings of the 22nd International Sympoisum on Fault-Tolerant Computing, Boston, Massachusetts, July 8–10, 1992, pp. 475–484 (1992).
14. Zheng J., Williams L., Nagappan N., Snipes W., Hudepohl J. P., Vouk M. A.: On the value of static analysis for fault detection in software. IEEE Transactions on Software Engineering, 32(4), 240–253 (2006).